REMARKS/ARGUMENTS

Claims 1-6, 9 and 11-15 stand rejected, with claims 7 and 8 objected to in the outstanding Official Action. Claims 14 and 15 have been amended and therefore claims 1-9 and 11-15 remain in the application.

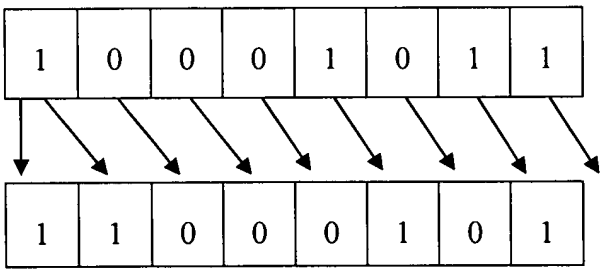
In sections 4 and 5 on page 2 of the outstanding Official Action, claims 14 and 15 stand rejected under 35 USC §101 as allegedly being directed to non-statutory subject matter. As noted in the attached Examiner initiated interview summary, the Examiner contacted Applicant's undersigned representative on August 17, 2006 and agreed that he would, by Examiner's Amendment, amend claims 14 and 15 in the manner of the above amendment. As noted in the continuation of the Examiner initiated interview summary, apparently the Examiner had second thoughts on the indicated allowance of the application and never made the agreed-upon amendments to claims 14 and 15. Applicant makes those amendments above which, as noted in the Official Action, obviates any further rejection to claims 14 and 15.

Claims 1, 3-6, 9, 11, 12, 14 and 15 stand rejected under 35 USC §102 as being anticipated by Motorola (MC88110 – Second Generation RISC Microprocessor User's Manual). In section 8 of the outstanding Official Action, the Examiner states that "the Examiner is interpreting 'arithmetic right shift' as being a right shift of an arithmetic value." (Page 4, lines 13-15). The Examiner has simply misunderstood the meaning of the phrase "arithmetic right shift" as it would be interpreted by one having ordinary skill in the art.

As evidence that those of ordinary skill in this art would understand "arithmetic right shift" consistent with the definition set out above, Applicant includes herewith a photocopy of a portion of an online version of a standard textbook "The C Book" published in 1991 which

provides an adequate definition of the phrase "arithmetic right shift." The website provides a copyright notice and information with regard to when the book was published. The address for the web page is http://publications.gbdirect.co.uk/c_book/chapter2/expressions_and_arithmetic.html. The definition occurs in section 2.8.2.3 entitled "The bitwise operators." The definition states "a logical shift shifts zeros into the most significant bit positions; an arithmetic shift copies the current contents of the most significant bit back into itself."

Once skilled in the art would understand that an arithmetic right shift involves an operation that is similar to a logical right shift except that the sign bit (typically the left-most bit) is filled with the signed bit of the original (unshifted) number instead of by zeros. A logical right shift shifts bits to the right and the bits which fall off the end of the word are discarded and the word is filled with zeros from the left.



In the illustration of the arithmetic right shift above, note that the left-most bit is a 1 which is the sign bit. This sign bit is preserved while the right-most bit falls off the right-hand end of the register.

The Examiner directs Applicant's attention to Figure 5-5 of Motorola and alleges that this shows an arithmetic right shift, since it shows that a portion G1 is selected and shifted to the

right towards the least significant end of the output rD. However, the referenced portion of Motorola is the "ppack" operation which merely rearranges bits of the data word and does not replicate the sign bit. In view of the well known definition of an "arithmetic right shift" as explained above, it is clear that since the sign bit is not replicated, the ppack instruction of Motorola cannot involve an arithmetic right shift. Accordingly, claim 1 is clearly not anticipated by the Motorola reference.

Thus, in view of the above, Motorola by failing to teach Applicant's claimed instruction decoder which incorporates "an arithmetic shift by a right-shift amount specified as a shift operand within said instruction," Motorola fails to anticipate claim 1 or claims 2-9 and 11-15, as there is no disclosure of the claimed apparatus in claim 1 or the method step of claim 14 or the computer program logic of claim 15. Accordingly, any further rejection of claims 1-9 and 11-15 over the Motorola reference is respectfully traversed.

Claims 1, 2, 4, 5, 9, 11, 12, 14 and 15 stand rejected under 35 USC §102 as being anticipated by Digital Equipment Corporation (DEC) (publication entitled "VAX11 780 Architecture Handbook"). On page 7, section 19 (iii)(a) of the Official Action, the Examiner states that the sign field of the input data word of the EXTV instruction extends from one end of the input data word. This is incorrect, since the sign bit of this instruction is at a bit offset position plus size, which need not be at one end of the input data word.

The Examiner references page 7-18 of DEC as support for his position. However, it is noted that the hashing in the figure on page 7-18 of DEC is still part of the input data word, as can be seen from the instruction description. Quite clearly, the input data word is a single 32-bit data word and the hashing on both sides is part of the input data word.

Claim 1 specifies two separate data words Rn and Rm and thus if DEC teaches only a single input data word, it would not only fail to anticipate claim 1, but it would lead one of ordinary skill in the art away from Applicant's claim 1. Specifically, page 7-19 in DEC contains a discussion of the EXTV instruction having the format specified at the top of page 7-18, i.e., "opcode pos.rl, size.rb, base.vb, dst.wl." Looking at the first example on page 7-19, the example in terms of the format, specifies the opcode = EXTV, pos.rl = #5, size.rb = #10, base.vb = Work1 and dst.wl = R0. Clearly, there is only one variable input "Work1" since the other two values (#5 and #10) are "immediates." Thus, the input data word (base.vb) is "Work1" whereas the bits to extract are specified by the two immediate values #5 and #10. Execution of the EXTV instruction involves putting bits 5 through 14 of the 32-bit input data word Work1 into the destination register R0.

As a result, it is clear that DEC, on pages 7-18 and 7-19, teaches only a single word and therefore cannot possibly teach the two data words specified in independent claims 1, 14 and 15. How or why the Examiner believes DEC contains any disclosure of multiple data words, never mind the functional interrelationship and method steps set out in claims 1, 14 and 15, is simply not apparent from the Official Action. Accordingly, any further rejection of claims 1, 2, 4, 5, 9, 11, 12, 14 and 15 or any other claims dependent thereon under 35 USC §102 is respectfully traversed.

Claims 2 and 13 stand rejected under 35 USC §103 as being unpatentable over Motorola. As noted above, Motorola clearly does not anticipate claim 1, from which claims 2 and 13 ultimately depend. In fact, it was noted that Motorola teaches away from the "arithmetic right shift" and therefore would lead one of ordinary skill in the art away from the invention of

Applicant's independent claim 1 and claims 2 and 13 dependent thereon. Because Motorola teaches away from the claimed invention, it cannot support a rejection under 35 USC §103 and any further rejection thereunder is respectfully traversed.

Claim 13 stands rejected on page 10 of the Official Action under 35 USC §103 as being unpatentable over DEC. Again, claim 13 depends from claim 1 and the above disclosure that claim 1 is not anticipated or rendered obvious over the DEC reference is herein incorporated by reference. Because DEC teaches only a single word ("Work1" in the example), DEC clearly fails to teach the data words Rn and Rm or the interaction between those data words as set out in Applicant's independent claim 1 and claims dependent thereon. Accordingly, DEC not only fails to render obvious Applicant's claim 13, it would lead one of ordinary skill in the art away from Applicant's claim 1 and claim 13 dependent thereon. Accordingly, any further rejection of claim 13 over the DEC reference is respectfully traversed.

The Examiner's indication of allowable subject matter being included in claims 7 and 8 is very much appreciated, but in view of the above discussion, there is no reason for rewriting claims 7 and 8 in independent form at the present time.

The present invention is directed to the problem of increasing the efficiency with which a data processing system performs data processing operations. Additional data manipulation can be combined with a data packing operation such that one of the data portions to be combined in a packed output data word can be multiplied or divided by a power of two at the same time that it is being packed together with another data word portion. The additional data manipulation is particularly flexible, since the shift amount to which the second portion of the data word Rm is subjected is independent of the bit length A of a first portion of the other data word Rn with

which it has been combined to form the output data word. Neither Motorola nor DEC contain any recognition of the problem, let alone Applicant's unique solution to that problem by utilizing the claimed arithmetic right shift and the other interrelationships set forth between data words Rn and Rm. Accordingly, there is simply no basis for further rejection of Applicant's claims.

Having responded to all objections and rejections set forth in the outstanding Official Action, it is submitted that claims 1-9 and 11-15 are in condition for allowance and notice to that effect is respectfully solicited. In the event the Examiner is of the opinion that a brief telephone or personal interview will facilitate allowance of one or more of the above claims, he is respectfully requested to contact Applicant's undersigned representative.

Respectfully submitted,

NIXON & VANDERHYE P.C.

By: _____
Stanley C. Spooner
Reg. No. 27,393

SCS:kmm
901 North Glebe Road, 11th Floor
Arlington, VA 22203-1808
Telephone: (703) 816-4000
Facsimile: (703) 816-4100

<gbdirect>

Search [ ] Go

West Yorkshire Office

GBdirect Ltd
Bradford Design Exchange
34 Peckover Street
BRADFORD
BD1 5BD
West Yorkshire
United Kingdom

consulting@gbdirect.co.uk

Training: 0800 651 0338
General: +44 (0) 870 200 7273
Finance: +44 (0) 1353 615 174

South East Regional Office

GBdirect Ltd
18 Lynn Rd
ELY
CB6 1DA
Cambridgeshire
United Kingdom

The C Book — Disclaimer and Copyright Notice

The first edition of this book was based on a late draft of the ANSI standard for C and is copyright Mike Banahan. This online version is a reproduction of the second edition based on the published ANSI standard. The second edition was published in 1991, copyright Mike Banahan, Declan Brady and Mark Doran. By agreement with Declan Brady and Mark Doran, copyright in this online version and derived works is copyright Mike Banahan, 2003. The print versions were published by Addison Wesley.

This online version is derived from files in Unix nroff format discovered on a floppy disk just prior to a move of offices by GBdirect Ltd. It is believed that the files were were used by the publishers Addison Wesley in the preparation of the second print edition and that some amendments or corrections may have been made in the print version that are not reflected in this online version. The online version was prepared with the assistance of some Perl scripts written by Mike Banahan, by Steve King, who cleaned up the output of the Perl scripts and also by sterling work by Geoff Richards and Aaron Crane who performed magic with XSLT to produce the HTML documents.

The publication of the online version is for historical interest and readers are warned that it should be treated as an historical document. There is now a later standard for the C programming language and this publication cannot be considered current: whilst for the most part the current and the first standard are very

Printer-friendly version

The C Book

This book is published as a matter of historical interest. Please read the copyright and disclaimer information.

performed on the operands of the unary forms of the operators.

2.8.2.3. The bitwise operators

One of the great strengths of C is the way that it allows systems programmers to do what had, before the advent of C, always beer regarded as the province of the assembly code programmer. That of code was by definition highly non-portable. As C demonstrates, there isn't any magic about that sort of thing, and into the bargain i turns out to be surprisingly portable. What is it? It's what is often referred to as 'bit-twiddling'—the manipulation of individual bits in integer variables. None of the bitwise operators may be used on re operands because they aren't considered to have individual or accessible bits.

There are six bitwise operators, listed in Table 2.7, which also sho the arithmetic conversions that are applied.

Operator	Effect	Conversions
&	bitwise AND	usual arithmetic conversions
|	bitwise OR	usual arithmetic conversions
^	Bitwise XOR	usual arithmetic conversions
<<	left shift	integral promotions
>>	right shift	integral promotions
~	one's complement	integral promotions

Table 2.7. Bitwise operators

Only the last, the one's complement, is a unary operator. It inverts state of every bit in its operand and has the same effect as the una minus on a one's complement computer. Most modern computers work with two's complement, so it isn't a waste of time having it the

Illustrating the use of these operators is easier if we can use hexadecimal notation rather than decimal, so now is the time to se hexadecimal constants. Any number written with `0x` at its beginnin interpreted as hexadecimal; both `15` and `0xf` (or `0XF`) mean the sar thing. Try running this or, better still, try to predict what it does first then try running it.

```
#include <stdio.h>
#include <stdlib.h>

main(){
        int x,y;
        x = 0; y = ~0;

        while(x != y){
                printf("%x & %x = %x\n", x, 0xff, x&0xff);
                printf("%x | %x = %x\n", x, 0x10f, x|0x10f
                printf("%x ^ %x = %x\n", x, 0xf00f, x^0xf0
```

```
                printf("%x >> 2 = %x\n", x, x >> 2);
                printf("%x << 2 = %x\n", x, x << 2);
                x = (x << 1) | 1;
        }
        exit(EXIT_SUCCESS);
}
```

Example 2.9

The way that the loop works in that example is the first thing to stu
The controlling variable is `x`, which is initialized to zero. Every time
round the loop it is compared against `y`, which has been set to a w
length independent pattern of all `1`s by taking the one's compleme
zero. At the bottom of the loop, `x` is shifted left once and has 1 OR
into it, giving rise to a sequence that starts `0`, `1`, `11`, `111`, ... in binar

For each of the AND, OR, and XOR (exclusive OR) operators, `x` is
operated on by the operator and some other interesting operand, t
the result printed.

The left and right shift operators are in there too, giving a result wh
has the type and value of their left-hand operand shifted in the
required direction a number of places specified by their right-hand
operand; the type of both of the operands must be integral. Bits sh
off either end of the left operand simply disappear. Shifting by mor
bits than there are in a word gives an implementation dependent
result.

Shifting left guarantees to shift zeros into the low-order bits.

Right shift is fussier. Your implementation is allowed to choose
whether, when shifting signed operands, it performs a logical or
arithmetic right shift. This means that a logical shift shifts zeros int
the most significant bit positions; an arithmetic shift copies the curr
contents of the most significant bit back into itself. The position is
clearer if an unsigned operand is right shifted, because there is no
choice: it must be a logical shift. For that reason, whenever right sl
is being used, you would expect to find that the thing being shifted
been declared to be unsigned, or cast to unsigned for the shift, as
the example:

```
int i,j;
i = (unsigned)j >> 4;
```

The second (right-hand) operand of a shift operator does not have
be a constant; any integral expression is legal. Importantly, the rul
involving mixed types of operands do not apply to the shift operatc
The result of the shift has the same type as the thing that got shifte
(after the integral promotions), and depends on nothing else.

Now something different; one of those little tricks that C programm